Tutorial On RISC V

Simulator

- I'll be using Rars, also see description of system calls here. It as well have a nice companion documentation to understand more about risc v, it is expected that one has given it a read.
- Can try Venus, Github repo: https://github.com/kvakil/venus. Note that for system calls, their argument register is different, see this.
- Can use spike, installed when using `riscv-gnu-toolchain`. Note it was as well required to install pk. System calls are different than RARS, basically it follows linux system calls. Can see these system calls here1, and here2. And linux system calls here, note system calls of interest can be concisely seen here.

  So to compile and run the program, do: (Don't know if this is the intended way but after a lot of trial and error, I found this)

  ```
  riscv64-unknown-elf-as -o filename.o filename.s
  riscv64-unknown-elf-ld -o filename filename.o
  spike pk filename
  ```

Examples

Hello Word

```
.data

msg: .string "hello world"

.text

start:
  li a7, 4
  la a0, msg
  ecall
```

```
  li a7, 10
  ecall
```

To get the same code working using spike.

```
.globl _start # We must need to give _start, .globl helps to see it outside this file
.data # Tell the assembler we are defining data not code
str:  # Label this position in memory so it can be referred to in our code
  .string "Hello World!\n" # Copy the string "Hello World!\n" into memory

.text # Tell the assembler that we are writing code (text) now
_start: # Make a label to say where our program should start from

  li a0, 1   # li means to Load Immediate and we want to load the value 1 into register a0
  la a1, str # la is similar to li, but works for loading addresses
  li a2, 13  # like the first line, but with 13. This is the final argument to the system ca
  li a7, 64  # a7 is what determines which system call we are calling and we what to call wr
  ecall      # actually issue the call

  li a0, 0   # The exit code we will be returning is 0
  li a7, 93  # Again we need to indicate what system call we are making and this time we are
  ecall
```

Fibonacci

```
.globl _start

.data
  msg1: .string "Please enter a number: "
  msg2: .string "The "
  msg3: .string " fibonnaci number is: "

.text

_start:
  # Initial 2 fibs
  li t0, 0
  li t1, 1
  # prints msg1
  la a0, msg1
  li a7, 4
  ecall
  # reads an int and moves it to register t3
  li a7, 5
  ecall
  mv t3, a0
  # prints a newline character
  li a7, 11
  li a0, '\n'
  ecall
  # prints msg2
  la a0, msg2
  li a7, 4
  ecall
  # prints the int value in t3
  mv a0, t3
  li a7, 1
  ecall
  # fibonnaci program
fib:
  beq t3, zero, finish
  add t2, t1, t0
  mv t0, t1
  mv t1, t2
  addi t3, t3, -1
  j fib
finish:
  # prints msg3
  la a0, msg3
  li a7, 4
  ecall
  # prints the result in t0
  mv a0, t0
  li a7, 1
  ecall
  # prints a newline
  li a0, '\n'
  li a7, 11
  ecall
  # ends the program with status code 0
  li a7, 10
  ecall
```